TIE-20100 Tietorakenteet ja algoritmit, kevät 2018

Harjoitustyö 1: Kaupungit

Viimeksi päivitetty 31.01.2018

Harjoitustyön aihe

Tässä harjoitustyössä harjoitellaan yksinkertaisten algoritmien toteuttamista ja niiden tehokkuuden arvioimista.

Harjoitustyössä ohjelmaan syötetään tietoja kaupungeista (nimi ja koordinaatit), ja ohjelmalta voi kysyä kaupunkeja halutussa järjestyksessä sekä minimi- ja maksimitietoja. Ei-pakollisena osana kaupunkeja voi myös poistaa.

Koska kyseessä on Tietorakenteiden ja algoritmien harjoitustyö, ohjelman tehokkuus on tärkeä arvostelukriteeri. Tavoitteena on tehdä mahdollisimman tehokkas toteutus, kun oletetaan että kaikki ohjelman tuntemat komennot ovat suunnilleen yhtä yleisiä (ellei komentotaulukossa toisin mainita). Plussaa tietysti saa, mitä tehokkaammin operaatiot pystyy toteuttamaan.

Huomaa erityisesti seuraavat asiat:

- Valmiina annetun pääohjelman voi ajaa joko graafisen käyttöliittymän kanssa QtCreatorilla/qmakella käännettynä, tai tekstipohjaisena pelkällä g++:lla käännettynä. Itse ohjelman toiminnallisuus ja opiskelijan toteuttama osa on täsmälleen sama molemmissa tapauksissa.
- **Vihje** tehokkuudesta: Jos minkään operaation keskimääräinen tehokkuus on huonompi kuin $\Theta(n \log n)$, ratkaisun tehokkkuus *ei* ole hyvä. Suurin osa operaatioista on mahdollista toteuttaa paljon nopeamminkin.
- **Osana ohjelman palautusta tiedostoon datastructures.hh on jokaisen operaation oheen laitettu kommentti, johon lisätään oma arvio kunkin toteutetun operaation asymptoottisesta tehokkuudesta lyhyiden perusteluiden kera.**
- Operaatioiden `remove()` ja `towns_distance_increasing_from()` toteuttaminen ei ole pakollista läpipääsyn kannalta. Ne ovat kuitenkin osa arvostelua, joten toteuttamatta jättäminen vaikuttaa harjoitustyön arvosanaan!
- Tehokkuudessa olennaisinta on, miten ohjelman tehokkuus muuttuu datan kasvaessa, eivät pelkät sekuntimäärät.
- Operaation tehokkuuteen lasketaan kaikki siihen liittyvä työ, myös mahdollisesti alkioiden lisäyksen yhteydessä tehty operaation hyväksi liittyvä työ.
- Plussaa tietysti saa, jos operaatioita saa toteutettua vaadittua mahdollisimman tehokkasti.

- Samoin plussaa saa, mitä nopeammaksi operaatiot saa sekunteinakin (jos siis kertaluokka on vähintään vaadittu). **Mutta** plussaa saa vain tehokkuudesta, joka syntyy omista algoritmivalinnoista ja suunnittelusta. (Esim. kääntäjän optimointivipujen väänteley, rinnakkaisuuden käyttö, häkkerioptimoinnilla kellojaksojen viilaaminen eivät tuo pisteitä.)
- Riittävän huonolla toteutuksella työ voidaan hylätä.
- Esimerkkejä kysymyksistä, joilla tehokkuutta voi usein parantaa: Tehdäänkö jokin asia turhaan useaan kertaan? Voiko jonkin asian joskus jättää kokonaan tekemättä? Tehdäänkö joissain työtä enemmän kuin on välttämättä tarpeen? Voiko jonkin asian tehdä "lähes ilmaiseksi" samalla, kun tehdään jotain muuta?

Kaupunkien etäisyyksistä

Joissain operaatioissa vertaillaan kaupunkien etäisyyksiä (joko origosta (0,0) tai annetusta koordinaatista). Jotta automaattitesteissä ei jouduttaisi ongelmiin liukulukujen kanssa, sovitaan että tämän harjoitustyön maailmassa on mahdollista liikkua vain yksi yksikkö kerrallaan itään, länteen, pohjoiseen tai etelään (ei siis vinoon). Tällöin kahden pisteen välisen etäisyyden voi laskea yksinkertaisesti kaavalla $|x1-x2|+|y1-y2|$, eli koordinaattien erotusten itseisarvon summana (kiinnostuneille, tällaista geometriaa kutsutaan nimellä *Manhattan/taxicab geometry*).

Järjestämisestä

Kaupunkeja järjestettäessä on mahdollista, että etäisyyden mukaan järjestettäessä usealla on sama etäisyys (tai nimijärjestyksessä nimi). Tällaisten tapausten keskinäinen järjestys on mielivaltainen.

Ohjelman hyväksymissä nimissä voi olla vain kirjaimia A-Z ja a-z. Järjestämisen voi tehdä joko std::string-luokan vertailuoperaattorin "<" mukaan (jossa isot kirjaimet tulevat ennen pieniä) tai "oikealla tavalla", jossa vastaavat isot ja pienet kirjaimet ovat samanarvoisia.

Harjoitustyön toteuttamisesta ja C++:n käytöstä

Harjoitustyön kielenä on C++14. Ohjelmointikielten valmiita järjestämisalgoritmeja (esim. `std::sort`, `std::nth_element`, `std::list::sort` jne.) ***EI*** saa käyttää työn tekemisessä, vaan tarvittava järjestäminen yms. toteutetaan itse. Tietorakenteena saa käyttää vain kielten tarjoamia sarjarakenteita (`std::array`, `std::vector`, `std::deque`, `std::list`), mutta ***EI*** assosiatiivisia tietorakenteita (esim. `std::map`, `std::set`).

Ohjelman toiminta ja rakenne

Osa ohjelmasta tulee valmiina kurssin puolesta, osa toteutetaan itse.

Valmiit osat, jotka tarjotaan kurssin puolesta

Tiedostot *mainprogram.hh, mainprogram.cc, mainwindow.hh, mainwindow.cc, mainwindow.ui* (joihin **EI SAA TEHDÄ *MITÄÄN* MUUTOKSIA)**

- Pääohjelma, joka hoitaa syötteiden lukemisen, komentojen tulkitsemisen ja tulostusten tulostamisen. Pääohjelmassa on myös valmiina komentoja testaamista varten.

- QtCreatorilla tai qmakella käännettäessä graafinen käyttöliittymä, jonka "komentotulkkiin" voi näppäimistön lisäksi hiirellä lisätä komentoja, tiedostoja yms. Graafinen käyttöliittymä näyttää myös luodut kaupungit graafisesti samoin kuin suoritettujen operaatioiden tulokset.

Tiedosto *datastructures.hh*

- `class Datastructures`: Luokka, johon harjoitustyö kirjoitetaan. Luokasta annetaan valmiina sen julkinen rajapinta (johon **EI SAA TEHDÄ *MITÄÄN* MUUTOKSIA)**
- Tyyppimäärittely `TownData`, johon talletetaan kaupungin tiedot (ja jota käytetään operaatioiden paluuarvona luokan rajapinnassa).

Tiedosto *datastructures.cc*

- Tänne luonnollisesti kirjoitetaan luokan operaatioiden toteutukset.
- Funktio `random_in_range`: Arpoo luvun annetulla välillä (alku- ja loppuarvo ovat molemmat välissä mukana). Voit käyttää tätä funktiota, jos tarvitset toteutuksessasi satunnaislukuja.

Harjoitustyönä toteutettavat osat

Tiedostot *datastructures.hh* ja *datastructures.cc*

- `class Datastructures`: Luokan julkisen rajapinnan jäsenfunktiot tulee toteuttaa. Luokkaan saa listätä omia määrittelyitä (jäsenmuuttujat, uudet jäsenfunktiot yms.)
- **Tiedostoon *datastructures.hh* kirjoitetaan jokaisen toteutetun operaation yläpuolelle kommentteihin oma arvio ko. operaation toteutuksen asymptoottisesti tehokkuudesta ja lyhyt perustelu arviolle.**

Huom! Omassa koodissa ei ole tarpeen tehdä ohjelman varsinaiseen toimintaan liittyviä tulostuksia, koska pääohjelma hoitaa ne. Mahdolliset Debug-tulostukset kannattaa tehdä cerr-virtaan (tai `qDebug`:lla, jos käytät Qt:ta), jotta ne eivät sotke testejä.

Ohjelman tuntemat komennot ja luokan julkinen rajapinta

Kun ohjelma käynnistetään, se jää odottamaan komentoja, jotka on selitetty alla. Komennot, joiden yhteydessä mainitaan jäsenfunktio, kutsuvat ko. Datastructure-luokan operaatioita, jotka siis opiskelijat toteuttavat. Osa komennoista on taas toteutettu kokonaan kurssin puolesta pääohjelmassa.

Jos ohjelmalle antaa komentoriviltä tiedoston parametriksi, se lukee komennot ko. tiedostosta ja lopettaa sen jälkeen.

Komento **`Julkinen jäsenfunktio`**	**Selitys**
add_town nimi (x,y) **`TownData* add_town(std::string const& name, int x, int y);`**	Lisää tietorakenteeseen uuden kaupungin annetulla nimellä ja sijainnilla.

Komento `Julkinen jäsenfunktio`	Selitys
remove nimi `void remove_town(std::string const& town_name);`	Poistaa annetulla nimellä olevan kaupungin. Jos kaupunkia ei ole lisätty, ei tee mitään. Jos samalla nimellä on useita kaupunkeja, poistetaan jokin niistä. Tämän operaation tehokkuus ei ole kriittisen tärkeää (sitä ei oleteta kutsuttavan usein), joten se ei ole oletuksena mukana tehokkuustesteissä. *Tämän operaation toteuttaminen ei ole pakollista (mutta otetaan huomioon arvostelussa).*
size `unsigned int size();`	Palauttaa tietorakenteessa olevien kaupunkien lukumäärän.
clear `void clear();`	Tyhjentää tietorakenteet eli poistaa kaikki kaupungit (tämän jälkeen size palauttaa 0).
find nimi `TownData* find_town(std::string const& name);`	Palauttaa kaupungin, jolla on annettu nimi. Jos ko. nimistä kaupunkia ei ole, palauttaa nullptr:n. Jos ko. nimisiä kaupunkeja on useita, palautetaan jokin niistä.
all_towns `std::vector<TownData*> all_towns();`	Palauttaa kaikki tietorakenteessa olevat kaupungit mielivaltaisessa järjestyksessä. Tämä operaatio ei ole oletuksena mukana tehokkuustesteissä.
alphalist `std::vector<TownData*> towns_alphabetically();`	Palauttaa kaupungit nimen mukaan aakkosjärjestyksessä.
distlist `std::vector<TownData*> towns_distance_increasing();`	Palauttaa kaupungit etäisyysjärjestyksessä origosta (0,0), lähin ensin. *Huomaa etäisyyden määritelmä aiempana.*
mindist `TownData* min_distance();`	Palauttaa kaupungin, joka on lähinnä origoa (0,0). Jos tällaisia on useita, palauttaa jonkin niistä. *Huomaa etäisyyden määritelmä aiempana.*
maxdist `TownData* max_distance();`	Palauttaa kaupungin, joka on kauimpana origosta (0,0). Jos tällaisia on useita, palauttaa jonkin niistä. *Huomaa etäisyyden määritelmä aiempana.*
nth_distance n **TownData* nth_distance(unsigned int n);**	Palauttaa kaupungin, joka on etäisyysjärjestyksessä n:s origosta (0,0). Jos tällaisia on useita, palauttaa jonkin niistä. Jos *n* on 0 tai suurempi kuin kaupunkien määrä, palautetaan nullptr. *Huomaa etäisyyden määritelmä aiempana.*
towns_from (x,y) **std::vector<TownData*> towns_distance_increasing_from(int x, int y);**	Palauttaa kaupungit etäisyysjärjestyksessä annetusta koordinaatista (x,y), lähin ensin. *Huomaa etäisyyden määritelmä aiempana. Tämän operaation toteuttaminen ei ole pakollista (mutta otetaan huomioon arvostelussa).*
random_add n (pääohjelman toteuttama)	Lisää tietorakenteeseen (testausta varten) *n* kpl kaupunkeja, joilla on satunnainen nimi ja sijainti. Huom! Arvot ovat tosiaan satunnaisia, eli saattavat olla kerrasta toiseen eri.

Komento **`Julkinen jäsenfunktio`**	**Selitys**
random_seed n (pääohjelman toteuttama)	Asettaa pääohjelman satunnaislukugeneraattorille uuden siemenarvon. Oletuksena generaattori alustetaan joka kerta eri arvoon, eli satunnainen data on eri ajokerroilla erilaista. Siemenarvon asettamalla arvotun datan saa toistumaan samanlaisena kerrasta toiseen (voi olla hyödyllista debuggaamisessa).
read 'tiedostonimi' (pääohjelman toteuttama)	Lukee lisää komentoja annetusta tiedostosta. (Tällä voi esim. lukea listan tiedostossa olevia työntekijöitä tietorakenteeseen, ajaa valmiita testejä yms.)
stopwatch on / off / next (pääohjelman toteuttama)	Aloittaa tai lopettaa komentojen ajanmittauksen. Ohjelman alussa mittaus on pois päältä ("off"). Kun mittaus on päällä ("on"), tulostetaan jokaisen komennon jälkeen siihen kulunut aika. Vaihtoehto "next" kytkee mittauksen päälle vain seuraavan komennon ajaksi (kätevää read-komennon kanssa, kun halutaan mitata vain komentotiedoston kokonaisaika).
perftest all/compulsory/cmd1;cmd2... **timeout n n1;n2;n3...** (pääohjelman toteuttama)	Ajaa ohjelmalle tehokkuustestit. Tyhjentää tietorakenteen ja lisää sinne *n1* kpl satunnaisia kaupunkeja. Sen jälkeen arpoo *n* kertaa satunnaisen komennon. Mittaa ja tulostaa sekä lisäämiseen että komentoihin menneen ajan. Sen jälkeen sama toistetaan *n2*:lle jne. Jos jonkin testikierroksen suoritusaika ylittää *timeout* sekuntia, keskeytetään testien ajaminen (tämä ei välttämättä ole mikään ongelma, vaan mielivaltainen aikaraja). Jos ensimmäinen parametri on *all*, arvotaan lisäyksen jälkeen kaikkista komennoista, joita on ilmoitettu kutsuttavan usein. Jos se on *compulsory*, testataan vain komentoja, jotka on pakko toteuttaa. Jos parametri on lista komentoja, arvotaan komento näiden joukosta (tällöin kannattaa mukaan ottaa myös random_add, jotta lisäyksiä tulee myös testikierroksen aikana). Jos ohjelmaa ajaa graafisella käyttöliittymällä, "stop test" nappia painamalla testi keskeytetään (nappiin reagointi voi kestää hetken).
testread 'in-tiedostonimi' 'out-tiedostonimi' (pääohjelman toteuttama)	Ajaa toiminnallisuustestin ja vertailee tulostuksia. Lukee komennot tiedostosta in-tiedostonimi ja näyttää ohjelman tulostuksen rinnakkain tiedoston out-tiedostonimi sisällön kanssa. Jokainen eroava rivi merkitään kysymysmerkillä, ja lopuksi tulostetaan vielä tieto, oliko eroavia rivejä.
help (pääohjelman toteuttama)	Tulostaa listan tunnetuista komennoista.
quit (pääohjelman toteuttama)	Lopettaa ohjelman. (Tiedostosta luettaessa lopettaa vain ko. tiedoston lukemisen.)

"Datatiedostot"

Kätevin tapa testata ohjelmaa on luoda "datatiedostoja", jotka add-komennolla lisäävät joukon kaupunkeja ohjelmaan. Kaupungit voi sitten kätevästi lukea sisään tiedostosta read-komennolla ja sitten kokeilla muita komentoja ilman, että kaupungit täytyisi joka kerta syöttää sisään käsin.

Alla on esimerkki datatiedostosta, joka löytyy nimellä *example-data.txt*:

```
# Adding towns
add_town Helsinki (3,0)
add_town Tampere (2,2)
add_town Oulu (3,5)
add_town Kuopio (6,3)
```

Esimerkki ohjelman toiminnasta

Alla on esimerkki ohjelman toiminnasta. Esimerkin syöte löytyy tiedostosta *example-in.txt* ja alla oleva tulostus tiedostosta *example-out.txt*. Eli voit testata esimerkin toimimista käynnistämällä ohjelman ja antamalla komennon *testread 'example-in.txt' 'example-out.txt'*.

```
> clear
Cleared all towns
> size
Number of towns: 0
> read "example-data.txt"
** Commands from 'example-data.txt'
> # Adding towns
> add_town Helsinki (3,0)
Helsinki: pos=(3,0)
> add_town Tampere (2,2)
Tampere: pos=(2,2)
> add_town Oulu (3,5)
Oulu: pos=(3,5)
> add_town Kuopio (6,3)
Kuopio: pos=(6,3)
>
** End of commands from 'example-data.txt'
> size
Number of towns: 4
> alphalist
1. Helsinki: pos=(3,0)
2. Kuopio: pos=(6,3)
3. Oulu: pos=(3,5)
4. Tampere: pos=(2,2)
> mindist
Helsinki: pos=(3,0)
> maxdist
Kuopio: pos=(6,3)
> distlist
1. Helsinki: pos=(3,0)
2. Tampere: pos=(2,2)
3. Oulu: pos=(3,5)
4. Kuopio: pos=(6,3)
```

```
> nth_distance 2
Tampere: pos=(2,2)
> find Kuopio
Kuopio: pos=(6,3)
> towns_from (4,4)
1. Oulu: pos=(3,5)
2. Kuopio: pos=(6,3)
3. Tampere: pos=(2,2)
4. Helsinki: pos=(3,0)
> remove Kuopio
Kuopio removed.
> size
Number of towns: 3
```